VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

IDENTIFIKACE SÍŤOVÝCH APLIKACÍ ZE ŠIFROVANÉ KOMUNIKACE
IDENTIFICATION OF NETWORK APPLICATIONS FROM ENCRYPTED COMMUNICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RADIM ŠAFÁŘ**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. ONDŘEJ RYŠAVÝ, Ph.D.**
SUPERVISOR

BRNO 2024

Abstrakt

Cílem práce je vytvořit nástroj schopný detekovat aplikace ze šifrovaného provozu za pomocí strojového učení. Zdrojem dat pro klasifikaci jsou síťové toky získané z nástroje Suricata, ne celkový obsah komunikace. Hlavním zdrojem je TLS handshake, které jsme schopni otiskout pomocí otisků JA3 či JA4 a jednodušeji si tak spojení identifikovat. Práce částečně řeší i problematiku chybějící implementace otisku JA4 ve vybraných nástrojích. Pro klasifikaci je použita knihovna ML.NET, která velmi zjednodušuje celý proces vytváření modelu.

Abstract

The goal of this thesis is creation of tool that is able to detect applications from encrypted traffic using machine learning. Data source for clasification are network flows captured with tool Suricata, not the entire content of communication. Main source are TLS handshakes, which are able to be fingeprinted with fingerprints JA3 or JA4 making flows easier to identify. Thesis also addresses the issue of JA4 not being implemented in used tools. For clasification is used library ML.NET which makes the process of creating a model easier.

Klíčová slova

TLS, šifrované spojení, šifrovaná komunikace, klasifikace, JA4, JA3, webové prohlížeče

Keywords

TLS, encrypted traffic, encrypted communication, clasification, JA4, JA3, web browsers

Citace

ŠAFÁŘ, Radim. *Identifikace síťových aplikací ze šifrované komunikace*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Ondřej Ryšavý, Ph.D.

Identifikace síťových aplikací ze šifrované komunikace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Ondřeje Ryšavého, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Radim Šafář
9. května 2024

Poděkování

Chtěl bych poděkovat panu doc. Ryšavému za ochotu pravidelně konzultovat a za jeho rady, i když rozvržení mé práce nebylo zrovna ideální.

Obsah

1 Úvod **4**

2 Problematika **5**
2.1 Šifrovaná komunikace 5
2.1.1 SSL - Secure Sockets Layer 6
2.1.2 TLS - Transport Layer Security 7
2.1.3 QUIC 8

3 Analýza **9**
3.1 Zdroje dat 9
3.1.1 Datové toky 9
3.2 Možné identifikátory 10
3.3 Otisky spojení 10
3.3.1 JA3 10
3.3.2 JA4 11
3.4 Detail JA4 12
3.4.1 Vytváření 12
3.4.2 Implementace v nástrojích 13
3.4.3 JA4+ 13
3.5 Vybrané možnosti 14

4 Implementace a integrace **15**
4.1 Návrhy 15
4.2 Použité nástroje 16
4.2.1 Powershell 16
4.2.2 Suricata 16
4.2.3 Hyper-V 17
4.2.4 ML.NET 17
4.3 Modely ML.NET 18
4.3.1 FastForestOva 18
4.3.2 LightGbmMulti 18
4.4 Detail implementace 18
4.4.1 Sběr dat 19
4.4.2 Klasifikace 20
4.4.3 Výstup 21
4.5 Integrace do dostupných nástrojů 21
4.5.1 Suricata 21
4.5.2 Fluent Bit 21

4.5.3 ELK stack . . . 22
4.5.4 WSL . . . 22

5 Vyhodnocení **24**
5.1 Automatická tvorba datových sad . . . 24
5.2 Použitý model . . . 24
5.2.1 Aplikace . . . 25
5.3 Výsledky klasifikace . . . 25
5.3.1 Výsledky . . . 26

6 Závěr **30**

Literatura **31**

A Datová sada **34**

Seznam obrázků

2.1 Rozdíl mezi HTTP a HTTPS komunikací 5
2.2 SSL 3.0 Handshake 6
2.3 Ukázky TLS 1.2 a TLS 1.3 handshake 7

3.1 Data ze kterých je vytvářen JA3 11
3.2 Rozdělení JA4 12

4.1 Návrh implementace 15
4.2 Vizualizace pomocí nástroje Kibana 23

5.1 Klasifikace při používání prohlížeče Google Chrome 27
5.2 Klasifikace při používání prohlížeče Microsoft Edge 28
5.3 Klasifikace při používání prohlížeče Mozilla Firefox 28
5.4 Klasifikace při používání prohlížeče Opera 29
5.5 Klasifikace při použití dalších aplikací 29

Kapitola 1

Úvod

V dnešní době jde valná většina internetové komunikace šifrované pomocí TLS. V České republice se podíl spojení šifrované TLS pohybuje kolem 80%[21]. Z pohledu uživatele má šifrované spojení spoustu výhod, hlavně co se týče bezpečnosti. Třetí osoba sledující náš provoz nemůže získat mnoho informací o naší komunikaci, natož změnit obsah dat nebo se za nás vydávat.

Na druhou stranu, z pohledu administrátora, nám šifrování spojení dělá monitorování sítě složitější. Zatímco dříve bylo možné vidět obsah komunikace, dnes je zašifrovaný a nečitelný. Může se tak stát, že například ve firemním prostředí zaměstnanci využívají aplikace, které by neměli. Nebo v horším případě ani neví, že z jejich počítače komunikuje nějaká nechtěná aplikace. Snaha o získání dat ze šifrované komunikace není žádnou novinkou. Ať už jde o získání obsahu HTTP hlaviček[7] či zjištění prohlížeče [15].

Cílem mé práce je vytvoření aplikace, která zvládne detekovat aplikace ze šifrovaného provozu. Navazuji na práci doc. Ryšavého, který již experimentoval s použitím JA3 hashů pro identifikaci aplikací. Má práce je založena na zjišťování veškerých informací pouze z datových toků, které získává z nástroje Suricata. Informace o tocích následně kombinuje s informacemi aplikacích, které inicializovali spojení, na vytvoření datové sady. Tato datová sada je následně použita v knihovně strojového učení ML.NET k natrénování klasifikačního modelu.

Nástroj je integrován do nástroje Suricata nepřímo. Pomocí nástroje Fluent Bit je jeden z souborových výstupů přeposílán do klasifikačního programu. Ten následně výsledky posílá do nástroje Logstash, který je součástí ELK stacku. Nástroj Kibana je poté použit na grafické zobrazení výsledků.

Kapitola 2

Problematika

Dříve, když chtěl někdo sledovat síťový provoz, ať už z důvodu monitorování firemní sítě nebo hledání možnosti jak do sítě neoprávněně vniknout, to mohl provádět velmi jednoduše. Protokoly jako HTTP[22] nebo DNS[19][20] ve svém základu nepočítají se žádným šifrováním. Bylo tak možné vidět, co za data přesně přes síť prochází, pokud aplikace samotné nepoužívaly nějaké šifrování. Toto vedlo k relativně snadnému podvržení zpráv, falšování identity aj. Tyto důvody, a mnohé další, vedly k vytvoření protokolu SSL[11] a následně TLS[3]. Oba protokoly fungují jako obálky dalších dat, například HTTPS či DNS over TLS (DoT).

Rozdíl jak vypadá nešifrovaná a šifrovaná komunikace lze vidět na 2.1, kde je vedle sebe zachycená HTTP a HTTPS komunikace se stejnou webovou stránkou. U HTTP komunikace lze krásně vidět obsah jednotlivých zpráv (v tomto příkladu se jedná o obrázky webové stránky). V HTTPS pouze vidíme, že byla komunikace navázána, ale obsah samotné komunikace je šifrovaný.

```
Info
GET /img/AmmoUniformWIcon.png HTTP/1.1
HTTP/1.1 200 OK  (PNG)
GET /map_tiles/2/2_2_3.png HTTP/1.1
GET /map_tiles/2/2_1_3.png HTTP/1.1
GET /map_tiles/2/2_0_3.png HTTP/1.1
GET /map_tiles/2/2_3_3.png HTTP/1.1
HTTP/1.1 200 OK  (PNG)
HTTP/1.1 200 OK  (PNG)
GET /assets/panel-icons/MapIconStaticBase2.png HTTP/1.1
HTTP/1.1 200 OK  (PNG)
HTTP/1.1 200 OK  (PNG)
GET /assets/panel-icons/MapIconFactory.png HTTP/1.1
HTTP/1.1 200 OK  (PNG)
HTTP/1.1 200 OK  (PNG)
GET /assets/panel-icons/MapIconSalvage.png HTTP/1.1
GET /assets/panel-icons/Note.png HTTP/1.1
```

(a) HTTP komunikace

```
Info
Client Hello (SNI=hq.mreboy.com)
Server Hello, Change Cipher Spec, Application Data
Application Data, Application Data, Application Data
Change Cipher Spec, Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Application Data
Client Hello (SNI=maxcdn.bootstrapcdn.com)
Client Hello (SNI=use.fontawesome.com)
Client Hello (SNI=cdnjs.cloudflare.com)
Server Hello, Change Cipher Spec
```

(b) HTTPS komunikace

Obrázek 2.1: Rozdíl mezi HTTP a HTTPS komunikací

2.1 Šifrovaná komunikace

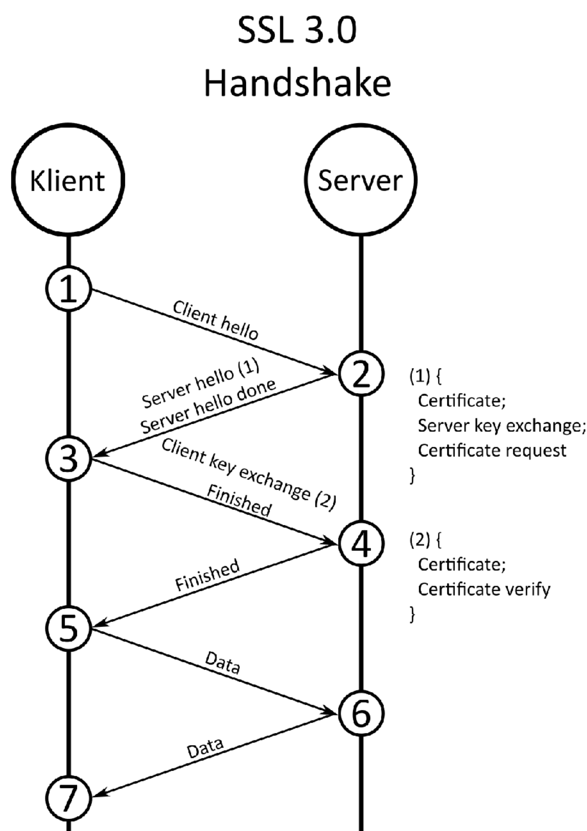
Šifrovaná komunikace má za úkol zajistit důvěrnost, autentizaci, integritu, neodmítnutelnost a dostupnost. Jeden z prvních protokolů zajišťující šifrovanou komunikaci bylo SSL 1.0 (Secure Sockets Layer) vytvořené firmou Netscape pro jejich webový prohlížeč[16]. Tento protokol prošel několika verzemi (1.0, 2.0 a 3.0) až byl nakonec nahrazen protokolem TLS 1.0

(Transport Layer Security). Ten byl následně aktualizován na verze 1.1, 1.2 a 1.3. V dnešní době je hlavně používaná verze 1.3[9]

2.1.1 SSL - Secure Sockets Layer

Technologie vytvořená původně pro účely Netscape Explorer v roce 1994[16], která využívá TCP. První implementace (SSL 1.0) nebyla nikdy zveřejněna. Další verze (SSL 2.0) přišla v 1995 a byla úpravou předchozí, primárně opravovala bezpečnostní chyby. Stále obsahovala několik velmi problematických chyb, například to byl problém s navázáním spojení, který dovoloval man-in-the-middle útok. Proto v roce 1996 přišla další verze (SSL 3.0), která protokol celý předělala. Hlavní výhodou bylo bezpečnější navazování spojení nebo detekce chyb. Tato verze je uveřejněna i jako RFC[11].

SSL 3.0 navazovalo spojení pomocí tzv. Handshake. Účel bylo domluvení společného symetrického klíče na šifrování dat pomocí asymetrické kryptografie. Obě strany si asymetricky vyměnili informace ke společnému symetrickému klíči. Handshake byl složitější než v TLS, protože obsahoval spoustu alternativ k předání klíče či certifikátu. Na obrázku 2.2 lze vidět jednu z možností. Zprávy uvedené bokem jsou buď volitelné nebo závislé na situaci, nejsou tedy posílány pokaždé. SSL 3.0 bohužel obsahuje bezpečnostní chyby, které dovolovali útoky jako POODLE[2]. Z tohoto důvodu se v dnešní době již SSL nedoporučuje používat.



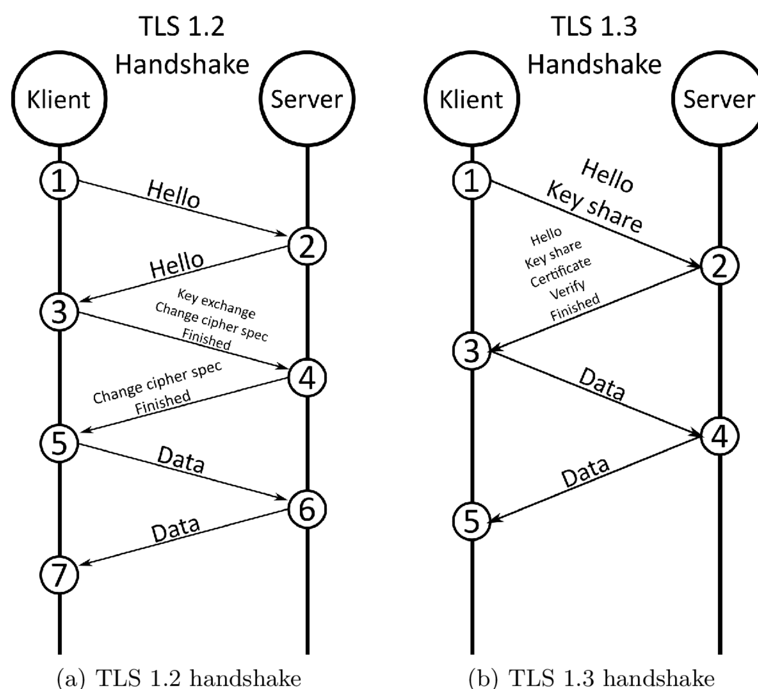
Obrázek 2.2: SSL 3.0 Handshake

Z důvodu původního rozšíření protokolu, se často můžeme i v dnešní době se zkratkou SSL potkat, i když používáno je pouze ve starých zařízení. Ve většině případů se tak ale

pouze nesprávně označuje protokol TLS. Občas lze najít i kombinaci zkratek, SSL/TLS, které také označuje TLS.

2.1.2 TLS - Transport Layer Security

Protokol TLS 1.0[3] vznikl jako vylepšení SSL 3.0 v roce 1999. Hlavní účel byla oprava bezpečnostních děr v SSL. V průběhu času se objevili problémy, jako útok BEAST[1], z tohoto důvodu byla vydána další verze (TLS 1.1[10]) v roce 2006. V průběhu let byla přidána podpora lepších algoritmů, proto v roce 2008 bylo vydáno TLS 1.2[24] podporující algoritmy jako SHA-256 a zároveň se odstranily již zastaralé možnosti. Kromě toho zlepšila navazování spojení, kde podporuje větší flexibilitu domluvy a tak bezpečnější konfigurace. Tato verze TLS je stále používána, i když domimantní[9] je TLS 1.3[23], která byla vydána v roce 2018. TLS 1.3 zjednodušilo celý proces navázání spojení tím, že zmenšilo počet posílaných zpráv. Zároveň přidalo podporu pro nové šifry a odstranilo podporu pro starší šifry.



(a) TLS 1.2 handshake (b) TLS 1.3 handshake

Obrázek 2.3: Ukázky TLS 1.2 a TLS 1.3 handshake

TLS navazuje spojení podobně jako SSL pomocí Handshake, jak lze vidět na obrázku 2.3. Není ale tak komplikovaný a nemá tolik alternativ. Klient pošle serveru zprávu Client Hello s dvěma náhodnými hodnotami. Původně se v Client Hello posílala v TLS 1.2 pouze jedna náhodná hodnota a druhá až po odpovědi Server Hello. Druhá hodnota je totiž zašifrována pomocí veřejného klíče serveru. Díky omezenému množství algoritmů na výměnu klíčů, si může klient dovolit poslat svoji Diffie-Hellman hodnotu už v Client Hello místo toho aby čekal na odpověď serveru, který algoritus chce použít. Server mu na to odpoví pomocí Server Hello ve které oznámí kterou šifru vybral, svůj certifikát a náhodnou hodnotu. Protože má všechny tři hodnoty na vypočítání již symetrického klíče, tak posílá i zprávu Finished. Klient si také následně spočítá symetrický klíč a může začít komunikovat.

2.1.3 QUIC

QUIC[13] je moderní protokol pro přenos dat přes internet. Je vyvíjený hlavně společností Google, ale je také standardizován pod IETF (Internet Engineering Task Force). Z tohoto důvodu se s ním můžeme potkat primárně v produktech Googlu.

Protokol má za cíl být rychlý, z tohoto důvodu používá multiplexování více datových toků v jenom spojení, což eliminuje nutnost vytváření vlastního TCP spojení pro každou komunikaci. Bezpečnost zajišťuje integrovanou šifrovací vrstvou pomocí protokolu TLS. Protokol se umí přizpůsobit podmínkám sítě pomocí detekce ztráty datagramů a následnou reakcí. I když je protokol postaven na technologii UDP, snaží se být co nejvíce podobný TCP pro snadnost použití. Protokol je používán protokolem HTTP/3, které zaručuje rychlejší komunikaci, lepší bezpečnost a 0-RTT obnovení spojení. Problematikou je ale právě postavení nad bezstavovým UDP, což se sebou nese problematiku bezstavového spojení, které je nutné vyřešit.

0-RTT (0 Round Trip Time) je mód QUIC, který dovoluje obnovení předchozího spojení bez nutnosti ho navazovat znovu. TLS 1.3 tuto možnost podporuje, bohužel se ale označení 0-RTT platí pouze pro samotné TLS, ale ne již o navázání TCP spojení. QUIC, protože pracuje na UDP spojení, dovoluje opravdové 0-RTT spojení. Klient, který již se serverem dříve komunikoval, může pouze znovu použít předchozí informace o spojení místo navazování úplně nového.

Navazování TLS spojení

TLS bylo navrhnuto pro stavové TCP spojení, QUIC ale používá UDP na přenos dat. TLS a QUIC se při navazování spolu doplňují[25]. Místo posílání zpráv TLS handshake přímo na server, se kterým se navazuje spojení, se zprávy posílají přes QUIC. Nejdříve QUIC naváže spojení se serverem, toto spojení následně využije TLS pro svůj handshake. Zprávy tedy nejsou posílány přes TLS Application Data záznamy, ale přes QUIC STREAM či jiné záznamy. Všechny jsou ale posílány přes QUIC pakety.

QUIC přenáší data pomocí CRYPTO záznamů, které obsahují kontinuální data TLS handshake. Tyto záznamy jsou zabaleny do QUIC paketů a poslány přes momentální úroveň šifrování, obvykle Initial. Z důvodu možné změny pořadí paketů má každý paket svůj typ, který obsahuje které klíče byly použity pro jeho zašifrování. Po dokončení TLS handshake si QUIC komponenta aktualizuje svoje šifrovací klíče a následně je používá.

Kapitola 3

Analýza

Pro identifikaci aplikaci síťových spojení je nutné najít charakteristické vlastnosti komunikace. V mém případě se jedná o atributy, které lze získat z šifrované komunikace. Navázal jsem na práci doc. Ryšavého, který již zkoušel propojit informace z TLS handshake k aplikacím. Pracoval s toky vybraných aplikací, jejich JA3 hash hodnotami, šiframi, rozšířeními a eliptickými křivkami. Popsal zde i problematiku identifikace prohlížeče Google Chrome, který vkládá GREASE hodnoty do TLS parametrů spojení. Z tohoto důvodu jsem se proto zabýval JA4, které kompenzuje nedostatky JA3.

3.1 Zdroje dat

Již ze začátku bylo třeba vybrat vhodný zdroj dat pro analýzu. Rozhodl jsem se pracovat s datovými toky, které jsou generovány například nástrojem Suricata, který jsem následně použil. Jeden z důvodu je zmenšení množství dat při zachování klíčových informací popisující komunikaci. Datové toky abstrahují reálná data na informace o zdroji, cíli, protokolu a jeho relevantních informací. Místo ukládání a zpracování kompletního záznamu TLS spojení se pracuje pouze se síťovými toky, což zjednodušuje extrakci relevantních dat. Síťové toky jsou k dispozici ve formátu JSON, který již obsahuje potřebné informace použitelné jako identifikátory aplikací.

3.1.1 Datové toky

Datové toky jsou jedním ze základních konceptů z oblasti síťové analýzy a správy sítí. Definují tok dat mezi zdrojem a cílem v počítačové síti a mohou být analyzovány či monitorovány za účelem řízení provozu, zabezpečení sítě a diagnostiky problémů.

Základními informacemi jsou zdroj a cíl, čas zahájení a ukončení, a charakteristika dat. Toky začínají na konkrétní zdrojové adrese a jdou na konkrétní cílovou adresu. Může to být například komunikace mezi klientem a serverem či i mezi klienty mezi sebou. Čas zahájení a ukončení toku nám říká, kdy komunikace probíhala. Mohou mí i svoji trvanlivost, po které se tok automaticky uzavírá a zda li je stále používán, vytváří se nový. Obsah informací v toku závisí na charakteristice dat. U DNS spojení nám může říct na jaký typ záznamu vedl dotaz či jakou odpověď jsme dostali.

3.2 Možné identifikátory

Z důvodu šifrované komunikace nemůžeme číst data jako takové. Ale i tak lze sledovat potřebné kroky k navázání spojení a následně spojení samotné. Moderní aplikace mají často definované doménové jméno na které se snaží připojit, to se následně přeloží na IP adresu, než IP adresu samotnou. Důvodem mohou být například CDN (Content Delivery Network). Musí se tedy použít DNS, které není šifrované a můžeme ho volně přečíst. Tohoto faktu využívají i programy sledující datové toky a logují tak DNS dotazy. Problémem může být DoT (DNS over TLS) nebo DoH (DNS over HTTPS), kde dotazy již jdou šifrovaně.

Dalším velmi silným identifikátorem je TLS spojení. Samotný přenos dat nám toho moc neřekne, protože je zašifrovaný. Jediné co z něho dokážeme zjistit jsou zdrojové a cílové adresy, případně velikost přenášených dat. Naštěstí TLS handshake není šifrovaný a z těchto čitelných informací můžeme vytvořit otisky, jak popisuji v další sekci. Otisky nám dovolí jednodušeji najít teoretické anomálie v provozu, protože se mohou rychleji zpracovat a porovnat, než celý tok. U TLS je logován právě handshake a následně množství přenesených dat. V závislosti na nástroji je ale detail logů TLS handshake různý a nemusí tak obsahovat vše potřebné pro výpočet otisku.

Třetím možným identifikátorem může být samotný datový tok. Informace o zdroji nám obvykle neposkytují významné údaje; IP adresa obvykle náleží klientovi a porty jsou systémem přidělovány dynamicky. Nicméně, potenciálně užitečné mohou být informace o cílové adrese. IP adresa serveru je často konstantní, což nám umožňuje teoreticky mapovat aplikace podle cílových IP adres. Ačkoliv toto mapování je možné i v případě portů, je to spíše vzácnost. Obvykle se setkáváme s portem 443 pro HTTPS, ale můžeme narazit i na jiné porty.

3.3 Otisky spojení

Pro rozlišení různých šifrovaných spojení lze efektivně využít metodu zvanou fingerprint (otisk). Na začátku každého TLS spojení dochází k procesu handshake, který probíhá v nešifrované formě. V této fázi jsou přenášeny informace jako verze protokolu TLS, nabízené šifrovací algoritmy a různá rozšíření s jejich hodnotami. Tyto informace mohou být využity k vytvoření kompaktního identifikačního řetězce pro každé spojení. Tato technika nám umožňuje zjednodušit monitorování síťového provozu tím, že místo kontroly všech parametrů spojení stačí sledovat výskyt nových otisků. Mezi používané otisky patří JA3. V září roku 2023 byla zveřejněna nová verze JA4.

3.3.1 JA3

JA3 byl zveřejněn v roce 2017[4]. Za jeho vznikem stojí trojice John Althouse, který následně vytvořil i JA4), Jeff Atkinson a Josh Atkins. Z tohoto důvodu se otisk jmenuje JA3, všichni tři jeho autoři mají iniciály JA.

Jak je vidět na obrázku 3.1, otisk je tvořen pomocí MD5 hashe z řetězce, který obsahuje verzi TLS, šifry, rozšíření, eliptické křivky a formát jejich bodů. Dekadické hodnoty těchto položek jsou následně spojeny pomlčkami a tyto nově vzniklé řetězce jsou následně dohromady spojeny čárkami. Z tohoto řetězce je následně spočítán MD5 hash, který je naším JA3 otiskem. Existuje i verze pro Server Hello nazvaná JA3S. JA3 hashe jsou například `b5001237acdf006056b409cc433726b0` nebo `2a596c42bf3e2e4f4b32f98b197480c3`.

```
Transport Layer Security
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 846
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 842
      Version: TLS 1.2 (0x0303)
      Random: 287945db8ff95a587cf145848976427397a2fedb31f27f05236b84a337844340
      Session ID Length: 32
      Session ID: 7a0460167f085e07152c1afae1fe66ed3658bb1fdee6fc3b8c194e9e5f6b66b6
      Cipher Suites Length: 32
      Cipher Suites (16 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 737
      Extension: Reserved (GREASE) (len=0)
      Extension: server_name (len=28)
      Extension: Unknown type 65037 (len=250)
      Extension: session_ticket (len=0)
      Extension: supported_versions (len=7)
      Extension: signed_certificate_timestamp (len=0)
      Extension: application_layer_protocol_negotiation (len=14)
      Extension: status_request (len=5)
      Extension: psk_key_exchange_modes (len=2)
      Extension: compress_certificate (len=3)
      Extension: supported_groups (len=10)
      Extension: application_settings (len=5)
      Extension: ec_point_formats (len=2)
      Extension: extended_master_secret (len=0)
      Extension: signature_algorithms (len=18)
      Extension: renegotiation_info (len=1)
      Extension: key_share (len=43)
      Extension: Reserved (GREASE) (len=1)
      Extension: pre_shared_key (len=272)
```

verze TLS
Šifry
Rozšíření
Křivky
Body

Obrázek 3.1: Data ze kterých je vytvářen JA3

V roce 2016 společnost Google vyvinula protokol GREASE (Generate Random Extensions and Sustain Extensibility), jehož účelem bylo zajištění správné implementace a rozšiřitelnosti protokolů podle RFC 8701 [8]. GREASE fungoval tak, že do procesu navazování spojení vkládal jednu z předem definovaných hodnot, které správně implementované systémy ignorovaly, zatímco systémy s chybnou implementací mohly na tyto hodnoty reagovat nedefinovaným způsobem. GREASE mohl vkládat předem definované hodnoty do šifer, rozšíření nebo eliptických křivek, což ovlivňovalo hodnoty JA3.

To vedlo k situaci, kde aplikace, které GREASE nepodporovaly, generovaly stále stejné JA3 hashe, zatímco aplikace implementující GREASE produkovaly pro každé spojení unikátní hash. Problém spočíval v metodě výpočtu hashe, která brala v úvahu všechny hodnoty, včetně náhodných přídavků od GREASE, a také neseřazeně, což znamenalo, že i pouhá změna pořadí rozšíření mohla změnit výsledný hash. JA3 se tak stal nepoužitelný pro detekci anomálií v síti, každé spojení s GREASE bylo vlastní anomálií.

3.3.2 JA4

V roce 2023 byla zveřejněna JA4[5]. Oproti JA3 má několik zásadních rozdílů, i když účel je stejný. Zatímco JA3 tvoří jeden hash ze všeho, JA4 je tvořeno ze tří různých částí. První část nám dává obecné informace o protokolu - tcp/quic, verze, počet šifer, počet rozšíření a ALPN. Druhá je SHA256 hash z seřazených šifer bez GREASE. Třetí je SHA256 hash z seřazených rozšíření bez ALPN a algoritmy podpisů také bez GREASE. Rozdíly mezi JA3 a JA4 lze vidět v tabulce 3.1. Pro hlavní část práce jsem si vybral právě otisk JA4, proto mu budu věnovat následující část.

JA3	JA4
2f2dc43000588c407270db99bef8c7eb	t13d1517h2_8daaf6152771_b0da82dd1658
c8cadafc142aa796dd73ad1d683bf916	t13d1517h2_8daaf6152771_b0da82dd1658
99c1a4712e8e0a79a481fb87ba1986da	t13d1516h2_8daaf6152771_02713d6af862

Tabulka 3.1: JA3 a JA4 hashe Client Hello

3.4 Detail JA4

JA4 je jeden otisk z větší kolekce nazvané JA4+[5]. Používá se na otisknutí Client Hello zprávy u TLS handshake. Existuje i varianta JA4S pro Server Hello. Pro účely práce ale nebyla použita. JA4 bylo zvoleno místo JA3, protože generuje produkuje lepší hodnoty nezávisle na GREASE. JA3 je jeden hash, zatímco JA4 je rozděleno na tři části, jak je zmíněno v předchozí části. Protože ignoruje GREASE, první dvě části otisku jsou obvykle pro aplikaci pokaždé stejné. Běžně používané aplikace nebudou mít potřebu se nějak anonymizovat například změnou šifer atd.

3.4.1 Vytváření

JA4 má tři části oddělené podtržítkem[6]. Jsou pojmenované JA4_a JA4_b a JA4_c, jak lze vidět na obrázku 3.2. JA4_a je složená část z:

- t pro TCP spojení, q pro QUIC spojení
- verze TLS/SSL (13, 12, 11, 10 pro TLS 1.3, 1.2, 1.1, 1.0 a s3, s2, s1 pro SSL 3.0, 2.0 a 1.0)
- d pokud jde zpráva na doménu (existuje sni rozšíření) nebo i na IP (není sni)
- dvouciferný počet šifer bez GREASE (např. 06), pro větší než 99 je počet 99
- počet rozšíření bez ALPN a GREASE
- první ALPN (Application-Layer Protocol Negotiation) hodnota (00 pokud neexistuje)

JA4_b (cipher hash) je následně SHA256 hash z seřazených hexadecimálních čísel šifer bez GREASE hodnot, oddělených čárkami, ze kterého je převzato prvních dvanáct znaků. JA4_c (extension hash) je také prvních dvanáct znaků z SHA256 hashe seřazených hexadecimálních čísel rozšíření oddělenými čárkami, který je spojený podtržítkem se seznamem čísel podpisových algoritmů v pořadí, ve kterém jsou poslány, teké oddělený čárkami.

t13d1517h2_8daaf6152771_b0da82dd1658
JA4_a JA4_b JA4_c

Obrázek 3.2: Rozdělení JA4

Příklad JA4_a

`t13d1517h2`

- t - TCP spojení

- 13 - TLS 1.3
- d - spojení šlo na doménu (existuje rozšíření SNI)
- 15 - Klient poslal 15 šifer bez GREASE hodnot
- 17 - Klient poslal 17 rozšíření bez GREASE hodnot
- h2 - první hodnota ALPN byla h2 (HTTP/2 over TLS)

Příklad JA4_b

Šifry: `8a8a,1301,1302,1303,c02b,c02f,c02c,c030,cca9,cca8,c013,c014,009c,009d,002f,0035`
Odstraníme GREASE hodnotu `8a8a`
Seřadíme: `002f,0035,009c,009d,1301,1302,1303,c013,c014,c02b,c02c,c02f,c030,cca8,cca9`
Vytvoříme SHA256 hash:
`8daaf6152771e33e12d734f9bc6478ed341f16cde27aee3aa36f2402f2c53b44`
A z něho vezmeme prvních 12 znaků: `8daaf6152771`

Příklad JA4_c

Rozšíření: `baba,fe0d,0023,002b,0012,0010,0005,002d,001b,000a,4469,000b,0017,000d,ff01,0033,eaea,0029`
Odstraníme SNI 0000, ALPN 000a a GREASE `baba` + `eaea`
Seřadíme: `0005,000a,000b,000d,0012,0017,001b,0023,0029,002b,002d,0033,4469,fe0d,ff01`
Vezmeme čísla podpisových algoritmů v pořadí, jak jsou poslány:
`0403,0804,0401,0503,0805,0501,0806,0601`
Oba seznamy spojíme podtržítkem: `0005,000a,000b,000d,0012,0017,001b,0023,0029,002b,002d,0033,4469,fe0d,ff01_0403,0804,0401,0503,0805,0501,0806,0601`
Z tohoto vytvoříme SHA256 hash:
`b0da82dd1658d3c9b45fe22e199326268ecdeb9e0163954e858ff5078b8b2970`
A ořízneme na prvních dvanáct znaků: `b0da82dd1658`

3.4.2 Implementace v nástrojích

Z důvodu že JA4 je relativně nově zveřejněn, nemá nejlepší rozšíření v nástrojích. Autoři přidali pluginy pro podporu do nástrojů Wireshark, Zeek a Arkime, s dalším rozšířením v budoucnu. Bohužel, v nástroji který jsem pro tuto práci použil, Suricata, podpora není. Dle autorů chybí platforma pro pluginy a nemají dostatek zdrojů ji do programu samostatně přidat. Podpora JA4 v Suricata je již implementována, bohužel první verze, která ji má obsahovat (8.0.0-beta1), bude vydána až 4. 6. 2024. Naštěstí se dá dobrá část JA4 vytvořit z informací co mám z řetězce na vytvoření JA3, tohoto faktu využívám v implementaci.

3.4.3 JA4+

V rodině JA4+ je několik dalších otisků, každé má trošku jiné využití.

Jeden z nich je JA4S, který je ekvivalent JA4 pro Server Hello. JA4S je vytvářen tak, že pro TLS Handshake generující JA4 hash `a_b_c` pokaždé bude JA4S hash `d_e_f`. Rozdílem

oproti JA4 je, že v jeho první části není počet šifer, server odpovídá pouze jednou, a v druhé části se nachází vybraná šifra. Ve třetí části chybí podpisové algoritmy, je to tedy pouze prvních dvanáct znaku SHA256 hashe ze seřazených rozšíření. Kombinace JA4 a JA4S nám pomůže identifikovat nejen knihovnu, kterou klient používá, ale i klienta samotného či o kterou skupinu malware se jedná.

Varianta JA4H je otisk pro HTTP spojení, obsahující HTTP metodu, verzi, existenci cookies a odkazovatele, počet http hlaviček a dalších informací. I když se může zdát, že není potřeba mít otisk na HTTP, když valná většina dat jde přes HTTPS, tak JA4H má využití na místech, kde není TLS využíváno. Některé malwary nepoužívají TLS a tak nejdou otiskout pomocí JA4.

Další verze jsou například JA4L, které otiskuje vzdálenost serveru od klienta, JA4X, způsob jakým jsou generovány TLS certifikáty, a JA4SSH pro SSH komunikaci

3.5 Vybrané možnosti

Po větším zkoumání možností identifikace jsem došel k tomu, že použiji primárně TLS a otisky jeho Handshake. Každá aplikace využívá nějakou TLS knihovnu, která předurčuje jaké používá šifry, rozšíření, eliptické křivky atd. Pomocí otisků jsme schopni tyto informace dát dohromady a rozhodnout, která aplikace spojení vytvořila. Častokrát se objevuje i rozšíření SNI (Server Name Identification), která nám dovoluje nepoužívat DNS jako takové. Díky použití JA4, se také vyhneme problému s GREASE, které měla JA3. Nevýhodou tohoto řešení je nutnost zachytit začátek celého toku. Otisk totiž vzniká z Handshake.

DNS jsem se rozhodl nepoužít, nejen, že například prohlížeče se budou připojovat na vícero adres, tak DNS jsou pouze dotazy. Spojení jsem schopen pro implementaci sledovat a párovat je s momentálně běžícími aplikacemi, u DNS toto nejde. Bylo by tedy nutné nejen sledovat DNS dotazy, ale i pomocí IP adres je následně párovat s TLS spojeními. Zároveň jsem schopen částečně nahradit toto složité párování právě zmínným rozšířením SNI, které se objevuje velmi často v TLS handshake.

Zároveň jsem přímo nepoužil datové toky (flow/netflow). Důvodem je problematika párování k aplikacím. Samotné IP adresy by totiž bylo potřeba nějakým způsobem propojit s aplikacemi. To by znamenalo vytváření databáze IP adres a aplikací, které se tam připojují. Výhodou je, že nemusím ale chytnout začátek toku, ale stačí mi informace, že existuje.

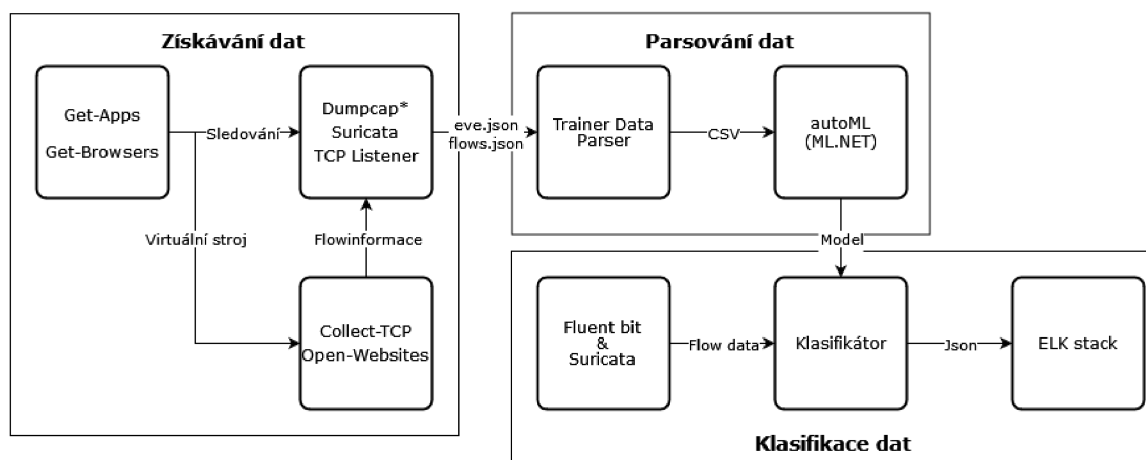
Kapitola 4

Implementace a integrace

Implementace programu pro detekci aplikací se dělí na několik částí. První část zajišťuje sběr informací pro vytvoření datových sad. Druhá vytváření datových sad samotných. Třetí pracuje se strojovým učením a klasifikuje vstupní informace. A poslední část je integrace do dostupných nástrojů, přesněji přijímání a odesílání dat.

4.1 Návrhy

Návrh implementace vytvořil výše zmíněné části. Bylo jasné, že než vytvářet jeden monolitický program, který obstará všechno, je lepší vytvořit několik menších. Grafický návrh lze vidět na obrázku 4.1.



Obrázek 4.1: Návrh implementace

Data byla získávána pomocí Powershell skriptu, který pouštěl nástroje na lokálním systému a virtuální stroji. Na lokálním stroji se data sbírala nástrojem Suricata a přijímáním dat, které posílal skript z virtuálního stroje. Na virtuálním stroji se pustil skript který detekoval připojení a pároval je s jménem aplikace, která spojení inicializovala. Obsahoval také další informace o toku samotném. Tento skript následně posílal výstup na server lokálního stroje. Ten tyto data postupně spojoval a vytvářel jeden výstupní soubor se všemi toky. Následně buď pustil aplikaci nebo v případě browseru začal otvírat webové stránky.

Následně byly veškeré tyto výstupy zkombinovány pomocí C sharp programu do datové sady vhodné pro strojové učení. Program počítal i část JA4 z JA3 string (JA3 ještě před

hašováním). Nebyl jsem schopen spočítat celou JA4, ale pouze JA4_a (až na ALPN hodnotu) a JA4_b. JA4_c je počítáno i pomocí neseřazených podpisových algoritmů. Z tohoto důvodu je často tento hash různý i pro stejnou aplikaci. Dle mého názoru tedy není třeba zkoumat jiné možnosti získání veškerých potřebných informací pro výpočet této části JA4. Jsem schopen získat relevantní části JA4 pouze z JA4 string.

Takto vytvořená datová sada byla použita pro strojové učení pomocí ML.NET. Výsledný klasifikační model byl využit dalším programem. Ten přijímal data přes TCP spojení a po klasifikaci odesílal data ve formátu json do dalšího nástroje přes HTTP.

4.2 Použité nástroje

Protože jsem čerpal z práce doc. Ryšavého, která byla napsána v Powershell, tak jsem se držel této technologie. Z tohoto důvodu jsem se držel operačního systému Windows. Toto rozhodnutí vedlo k využití Hyper-V jako virtualizačního nástroje. Pro sběr dat byl použit již zmiňovaný nástroj Suricata. Klasifikační model byl vytvořen pomocí nástroje ML.NET.

4.2.1 Powershell

Powershell je iteraktivní příkazový řádek a skriptovací jazyk vyvinutý společností Microsoft. Jedná se o nástroj pro automatizaci správy a konfigurace operačního systému Windows, aplikací a služeb. Pomocí příkazového řádku může uživatel pouštět příkazy přímo bez nutnosti psaní předchozího kódu. Lze tak manipulovat se soubory, zjišťovat informace o procesech a další jednorázové úkony.

Pro složitější operace je lepší napsat program, kde je Powershell využit jako skriptovací jazyk. Lze tak jednoduše automatizovat opakující se úlohy nebo zjednodušit pouštění složitejších serií příkazů. Zároveň je objektově orientovaný, lze tak vytvářet a využívat složitější datové struktury. Jednu z hlavních výhod je podpora modulů, které mohou přidat specifickou funkcionalitu. Ať už se jedná například o využití Powershell pro konfigurace AzureAD nebo ovládání virtualizační platformy Hyper-V. Toto mi dovolilo automaticky obnovovat virtuální stroj na vybraný kontrolní bod a pouštění skriptů či aplikací přímo ve virtuálním stroji. Powershell je velmi provázaný se systémem Windows, i když jeho poslední verze podporují i linuxové operační systémy či macOS. Zároveň je plně kompatibilní s .NET Framework, který zvládá plně využívat. Obvykle se setkáte s Powershell 5, práci je používaná verze 7.

4.2.2 Suricata

Suricata je open-source nástroj pro monitorování sítě a detekci hrozeb, který je vyvíjen neziskovou organizací OISF (Open Information Security Foundation). Hlavním cílem je detekce anomálií a potenciálně škodlivých aktivit v počítačových sítích. Nástroj obsahuje několik důležitých funkcí jako analýzu síťového provozu, detekci proniknutí (IDS - Intrusion Detection System) a prevenci proniknutí (IPS - Intrusion Prevention System). Využívá rozsáhlého systému konfigurovatelných pravidel pro hlášení výstrah.

V mém případě primárně používám hlavně výstupu `eve.json` (Extensible Event Format). Je to výstupní soubor pro detekované události (events) a výstrahy (alerts) během analýzy síťového provozu. Záznamy, které se do souboru zaznamenávají, jsou konfigurovatelné. Ať už jaké události se zaznamenávají tak i které informace o události. Pro moji implementaci je využito možnosti logování TLS spojení. Pro TLS Handshake bylo logováno

i jeho SNI rozšíření, verze a ja3. Veškeré ostatní výstupy nebyly třeba a tak nebyly v použité konfiguraci. Nevýhoda byla nemožnost tento výstup přímo přeposlat, byl proto použit nástroj Fluent Bit.

4.2.3 Hyper-V

Hyper-V je virtualizační platforma pro systém Windows vyvinuta společností Microsoft. Hlavním účelem je vytváření a správa virtuálních strojů na serverech a počítačích s operačním systémem Windows. Nástroj disponuje mnoha funkcemi a možnostmi jako je virtualizace serverů pro lepší využití hardwarových zdrojů, izolace systémů pro lepší testování softwaru, správa zdrojů, sítě aj. Virtualizační systém zvládá automaticky přidělovat zdroje virtualizovaným systémům, díky tomu nejsou takovou zátěží na hostitelský počítač.

Internetové připojení virtuálních strojů lze nastavit několika způsoby pomocí virtuálních přepínačů. Lze je připojit pomocí externího, interního nebo privátního virtuálního přepínače. Externí přepínač se přímo připojuje přes síťový adaptér do venkovní sítě, interní se připojí přes hostovací systém pomocí NAT spojení. Privátní síť dovoluje komunikaci mezi virtuálními stroji připojené na stejný přepínač. Zároveň virtualizační systém dovoluje vytváření kontrolních bodů pro zpětné obnovení virtualizovaného systému do podoby ve které byl kontrolní bod vytvořen. Díky tomuto jsem mohl zaručit stejné prostředí pro opakovaný sběr dat. V průběhu byl systém a aplikace několikrát aktualizovány, kvůli tomu bylo potřeba pravidelnému tvoření kontrolních bodů.

Pro mé řešení byl vybrán operační systém Windows 11 for Education. Jedná se o ekvivalent Windows 11 Pro určenou pro školy[14]. Hlavním důvodem pro vybrání kombinace Hyper-V pro virtualizaci a Windows 11 jako virtualizovaného operačního systému byla jednoduchost automatizace pomocí Powershell. Modul Hyper-V pro Powershell mi dovolil automaticky zapínat virtuální stroje, aplikovat kontrolní body a i pouštět aplikace či další Powershell skripty uvnitř virtuálního stroje. Sběr dat byl tím pádem opakovatelný.

4.2.4 ML.NET

ML.NET je open-source multiplatformí framework vyvíjený společností Microsoft pro strojové učení a vytváření modelů. Je navržen tak, aby umožňoval vývojářům efektivně vytvářet modely strojového učení v jazycích C sharp nebo F sharp. Poskytuje jednoduché API a nástroje pro vytváření, trénování a následného využívání modelů. Obsahuje různé algoritmy pro klasifikaci, regresi, shlukování a další úlohy. ML.NET je optimalizován a integrován do frameworku .NET. Dále nabízí integraci s dalšími knihovnami strojového učení jako je například TensorFlow a je rozšířitelný pomocí vlastních komponent. Jeho součástí je i Model Builder, což je jednoduchý nástroj pro vytváření vlastní modelů, který používá AutoML (Automated Machine Learning).

AutoML

AutoML je technologie, která automatizuje vytváření nejlepšího modelu pro daný scénář strojového učení s minimální nebo žádnou lidskou intervencí. Technologie automatizuje celý proces vytváření a optimalizaci modelů od výběru vhodného algoritmu až po jeho ladění. Provádí několik důležitých kroků jako předzpracování dat, výběr vhodného algoritmu či souboru algoritmů a trénovaní s různými konfiguracemi. Během tohoto procesu mohou také ladit hyperparametry modelu, aby maximalizovali jeho výkonnost ná základě měřené metriky. Nakonec vybírá nejlepší model, či modely, pro koncové použití.

Výhodou je možnost vytváření kvalitních modelů bez hlubokých znalostí v problematice strojového učení. Umožňuje také rychlejší iteraci a experimentování s různými přístupy. Nevýhodou je nemožnost přesného ladění modelu, může se tak stát, že je třeba model trénovat několikrát, dokud nedosáhne námi požadované přesnosti. Uživatel neví, jaké změny byly přesně v jednotlivých modelech udělány. V tomto případě bych tuto technologii použil na výběr algoritmu a následné trénování bych již udělal zvlášť.

4.3 Modely ML.NET

Při experimentech s ML.NET a různými datovými sadami byly často jako nejpřesnější označeny dva algoritmy. I když cílem práce nebylo detailní nastudování algoritmů strojového učení, rozhodl jsem se v rychlosti popsat tyto algoritmy, a to FastForestOva a LightGbm-Multi.

4.3.1 FastForestOva

FastForest[27] je algoritmus navazující na RandomForest[12] a je používán na klasifikaci a regresi dat. Je rychlejší ale neztrácí na přesnosti, zároveň lze použít i na zařízeních s nižším výpočetním výkonem jako telefony. Algoritmus je založen na několika rozhodovacích stromech, každý z nich je trénován na náhodné podmnožině. Je tím pádem zajištěna rozmanitost mezi stromy a je sníženo riziko přetrénování. Každý strom vyhodnotí vstupní data a jejich výsledky jsou následně agregovány.

4.3.2 LightGbmMulti

LightGBM[17] je vyvinutý společností Microsoft. Jedná se o výkonný model, který používá gradient boosting k vytváření modelů na základě stromových struktur. Jeden z hlavních rysů je jeho schopnost rychle a efektivně zpracovávat velké datové sady.

Využívá leaf-wise (best-first) růst, při každém kroku přidávání nového stromu do modelu se snaží vybrat list, který minimalizuje chybu (gradient ztráty). Umožňuje to algoritmu dosáhnout vyšší přesnosti s menším počtem listů, což vede k rychlejšímu trénování modelu. Pro urychlení trénování používá histogramové algoritmy, oproti obvyklým algoritmům, které jsou založené na předřazených datech (pre-sort-based algorithms), Tyto algoritmy jsou velmi optimalizovatelné, což vede k menší paměťové potřebě a rychlejšímu učení.

V práci je použita varianta LightGBM Multiclass, která je optimalizována pro řešení problému klasifikace s více než dvěma třídami. Klasická verze LightGBM je schopna řešit pouze binární klasifikaci. Toto rozšíření nám dovoluje kategorizovat data do více než dvou skupin.

4.4 Detail implementace

V této sekci projdu přesnější popis implementace. Při vývoji jsem pracoval na operačním systému Windows kvůli použití výše vybraných technologií jako Hyper-V a Powershell. Z důvodu použití ML.NET jsem používal integrované vývojové prostředí Visual Studio 2022 v kombinaci s Visual Studio Code pro psaní Powershell skriptů.

4.4.1 Sběr dat

Sběr dat probíhal automatizovaně pomocí Powershell skriptu `get-apps.ps1` a v případě prohlížečů `get-browser.ps1`. Veškeré jménem uvedené proměnné jsou argumenty programu. Skript nejdříve načetl obsah souboru `$App_file`, ve kterém jsou ve formátu jsou uloženy aplikace a případně i jejich argumenty pro spuštění, v případě prohlížečů tento soubor není třeba. Pokud virtuální stroj se jménem `$Vm_name` nebyl zapnutý tak ho skript zapne a následně obnoví na kontrolní bod `$Checkpoint`. Po zjištění IP adresy internetového rozhraní `$Interface` a objektu s přístupovými údaji vytvořeného z uživatelského jména `$User` a hesla `$Password` přejde na hlavní část skriptu. Hlavní části obou variant se mírně liší, jsou tedy rozepsané níže.

Varianta aplikací

Pro všechny aplikace nalezené v souboru `$App_file` opakuje následující část. Nejdříve určí název složky, kam bude ukládat výstupní data, jméno aplikace a její případné argumenty.

Poté spustí úlohy pro Suricatu a naslouchací server `$Listener`, na který jsou posílány záznamy ze skriptu pouštěného na virtuálním stroji, který bude popsán níže. Po připojení se na virtuální stroj, kde jsou nainstalované veškeré aplikace, které chceme spouštět, se nejdříve nastaví možnost pouštět Powershell skripty. Následně je spuštěn skript `collect-tcp.ps1`, který sleduje aktivní TCP spojení a dohledává názvy aplikací, které tyto spojení vytvořily. Tyto informace následně s informacemi o lokální IP a portu, vzdálené IP a portu, a čase spojení posílá ve formátu json na naslouchací server, který je umístěn na výchozí bráně virtuálního stroje, tj. hostovací stroj. Po čekání `$Wait_time` sekund je znovu nahrán kontrolní bod a čeká se 30 sekund, než Suricata zapíše veškeré toky. Po ukončení úloh Suricaty a naslouchacího serveru je část skriptu, která pouští úlohy a virtuální stroj, opakována `$Times` krát.

Nakonec po projití všech vstupních aplikací je virtuální stroj vypnnut.

Varianta prohlížečů

Pro variantu prohlížečů se v samotném skriptu nic neopakuje. Spustí se úlohy Suricaty a naslouchacího serveru `$Listener`, ve virtuálním stroji skript sledující TCP spojení. Následně místo pouštění aplikací samotných, se spustí skript `open-websites.ps1`.

Nejdříve načte prvních 1000 webových stránek ze souboru `top-1m.txt`, tento soubor jsem přebral z práce doc. Ryšavého. Skript v nekonečné smyčce nejdříve vybere prohlížeč a následně operaci, buď `Open` ve 3/4 případů a nebo `Close` v poslední 1/4. Při vybrání operace `Open` vybere náhodou webovou stránku a ve vybraném prohlížeči ho otevře. Při vybrání operace `Close` vybraný prohlížeč ukončí. Poté čeká náhodně vybranou dobu v intervalu od `$MinWait` a `$MaxWait` sekund.

Hlavní skript běží po dobu `$Duration` sekund. Poté obnoví kontrolní bod `$Checkpoint`, vypne virtuální stroj. Počká 30 sekund pro logování Suricaty a ukončí úlohy Suricaty a naslouchacího serveru.

Výstupy

Výstupem obou skriptů jsou dva soubory, `eve.json` a `flows.json`. Pro variantu aplikací dva soubory pro každý cyklus. Soubor `eve.json`zachycené TLS spojení nástrojem Suricata, kde lze vidět informace o TLS handshake. Pro nás jsou relevantní informace na jakou vzdálenou adresu bylo spojení navazování a následně informace vyextrahované z TLS Handshake

jako SNI (doménové jméno serveru), verze a JA3. Druhý soubor `flows.json` je výstupem skriptu `collect-tcp.ps1`, ve kterém nalezneme který proces ve virtuálním stroji vytvořil dané TCP spojení.

4.4.2 Klasifikace

Pro klasifikaci dat je nejdříve nutno data upravit do formátu, které přijímá ML.NET. Je pro to využit program `TrainerDataParser` psaný v C sharp. Program má dva vstupní parametry, a to `-source` a `-dest`. V parametru `-source` očekává složku, ve které najde soubory `eve.json` a `flows.json`. Parameter `-dest` slouží pro určení složky, do které bude zapsán výstupní csv soubor. Hlavním účelem programu je zpracování dvou vstupních souborů formátu json do jednoho výstupního souboru csv. Ve výstupním souboru nalezneme relevantní data z `eve.json` otagovaná názvy aplikací z `flows.json` a část JA4 otisku daného spojení.

Počítání JA4

Program počítá částečně JA4_a a celou JA4_b. Bohužel pro výpočet JA4_c nedostává ze Suricaty veškeré potřebné informace. Tato pracovní podoba JA4 je vytvářena z JA3 string, což je řetězcová hodnota JA3 ještě před zahašováním. Tento řetězec obsahuje dekadické hodnoty verze TLS, šifer, rozšíření, eliptických křivek a formátů bodů eliptických křivek. Jednotlivé části jsou spojené pomlčkami a celkově jsou spojené čárkami. Program vezme tento řetězec a rozdělí ho přes oddělovací čárky.

JA4_a vytváří postupně, nejdříve podle TLS verze určí první tři znaky. Pokud je verze označena již v `eve.json`, použije se tato informace. Pokud je označena jako `UNDETERMINED`, tak se rozhoduje podle existence rozšíření s hodnotou 43 (`Supported Versions`). Jestli toto rozšíření existuje, předpokládáme TLS 1.3, jinak TLS 1.2. Při existenci rozšíření s hodnotou 0 má server, na který se připojujeme, doménu a vkládá tak `d`, při neexistenci vkládá `i`. Následně vloží počet šifer a rozšíření, druhé a třetí části JA3 řetězce, na dvě číslice. V případě počtu přesahující dvě cifry je vložena hodnota 99. Poslední dvě hodnoty jsou první hodnota ALPN, bohužel JA3 řetězec nám dovolí zjistit, jestli rozšíření je přítomno, ale už ne jaké má hodnoty, proto je nahrazeno `xx`.

V jeho druhé části se nachází dekadické hodnoty šifer, které převede do hexadecimální podoby. Následně je seřadí, odstraní veškeré GREASE hodnoty a následně je opět zřetězí pomocí čárky. Tento řetězec je následně zahašuje pomocí SHA256 a vezme z tohoto hashe prvních 12 znaků, naše JA4_b.

JA4_c nejsme schopni spočítat, protože je tvořeno z seřazených hexadecimálních hodnot šifer bez GREASE a neseřazených hodnot podpisových algoritmů. Řetězec obsahuje hodnoty rozšíření, se kterými jsem schopen vytvořit první část řetězce před zahašováním. Bohužel JA3 nepoužívá podpisové algoritmy a tak pro vytvoření správného haše chybí druhá část hašovaného řetězce. Z tohoto důvodu jsem se rozhodl celou tuto část nahradit také písmeny `x`, než ji nechávat spočítanou napůl.

Strojové učení

Výstup z programu je datový set v csv souboru, který obsahuje relevantní informace pro strojové učení. Vybrané informace jsou cílová adresa `Dest_ip`, zdrojový port `Src_port`, doménové jméno serveru na který se navazuje TLS spojení `Tls_sni`, verze TLS `Tls_version` a JA4 `Tls_ja4_a` a `Tls_ja4_b`. Pro účely učení je přidán i sloupec `Tag`, ve kterém se nachází název aplikace, která spojení zahájila.

Tento datový set je následně využit v prostředí ML.NET na vytvoření modelu. Bylo vybrána možnost klasifikace dat, následně se zvolí prostředí a nahrají data, v našem případě vytvořený otagovanou datovou sadou. Po vybrání doby tréningu stačí pouze kliknout na `Start training` a počkat. ML.NET za stanovený čas trénování projde několik modelů a zjistí jejich přesnost. Díky rychlosti trénování bylo možné relativně snadno experimentovat.

Bohužel, ML.NET častokrát ukazuje různé přesnosti pro stejný dataset s využitím stejného algoritmu. Kvůli tomu, že dělá automatické úpravy v pozadí, tak nemůžeme zaručit identické výsledky, jak lze vidět v tabulce 4.1. Modely byly trénovány se stejnou datovou sadou, která obsahovala 2453 záznamů, po dobu dvou minut. Časy tréningu jednotlivých modelů se mohou lišit, zároveň někdy mohou být prozkoumány různé algoritmy, při neprozkoumání algoritmu je v tabulce místo hodnoty písmeno `X`. Celý proces tréningu je automatizovaný s možností nastavení metriky přesnosti, použitých algoritmů, ladiček a doby tréningu.

Název algoritmu	Micro přesnosti		
FastForestOva	0,7962	X	0,7962
FastTreeOva	0,7948	0,7948	0,7948
LightGbmMulti	X	0,6033	0,6033
LbfgsMaximumEntropyMulti	X	0,2019	X
LbfgsLogisticRegressionOva	X	0,2019	X

Tabulka 4.1: ML.NET přesnosti

4.4.3 Výstup

Výstupní program využívá natrénovaný model pro klasifikaci příchozích dat. Data přijímá ve formátu json na nastaveném portu. Následně je přetransformuje na validní vstupní data do modelu, pomocí kterého předpoví, jaká aplikace mohla stát za daným spojením. Výsledek klasifikace následně odesílá pomocí HTTP na cílové úložiště. Přesný způsob integrace v mém řešení je popsán v následující sekci.

4.5 Integrace do dostupných nástrojů

4.5.1 Suricata

Nástroj Suricata bohužel nepodporuje možnost pluginů ani možnost přímého odesílání dat na specifikovanou síťovou adresu. Bylo tedy nutné pracovat s výstupem `eve.json` jako se souborem, který je třeba dalším nástrojem odesílat dále, tato problematika je popsána v další podsekci. Moje práce neřeší problematiku dlouhodobého běhu tohoto programu. Neřeší velikosti souboru ani další druhy logů kromě TLS. Při použití je nutné tyto omezení nezanedbat.

4.5.2 Fluent Bit

Na odesílání dat ze souboru `eve.json` byl použit nástroj Fluent Bit. Open-source nástroj pro sběr, filtrování a odesílání logů a dalších datových toků. Má modulární architekturu, umožňující rozšíření pomocí pluginů pro různé účely jako jsou vstupy, filtry a výstupy.

Je navržen s ohledem na efektivitu a výkon, minimalizuje nároky na zdroje ale zároveň poskytuje spolehlivé zpracování dat.

Výběr byl na doporučení pana doc. Ryšavého, pro účely jednoduchého posílání změn v souboru. Na konfiguraci jsou použity dva soubory, `parsers.conf`, kde je vlastní parser pro `eve.json`, a `fluent_bit.conf`, kde je konfigurace samotného programu pro spuštění. Nachází se v něm konfigurace vstupu, kde je zapnut `Tail` vstup, který čte nové informace ze souborů, které dostal, a parser daného vstupu, v našem případě náš vlastní. Konfigurován je i výstup, který odesíláme ve formátu json na `localhost:5170`, kde si ho již přebírá program na klasifikaci dat.

Alternativou může být například program Filebeat. Program, který je vyvíjen společností Elastic, který je přímo dělaný na posílání dat do nástroje Logstash.

4.5.3 ELK stack

Výstup klasifikačního programu jde do programu Logstash, který je součástí kombinace technologií nazývaná ELK stack. ELK stack je kombinace Logstash, Elasticsearch a Kibana, všechny programy jsou open-surce a vyvíjené společností Elastic. Společně jsou určené pro sběr, ukládání, vizualizaci a analýzu datových záznamů. Pro moji práci slouží hlavně jako vizualizační nástroj pro klasifikovaná data.

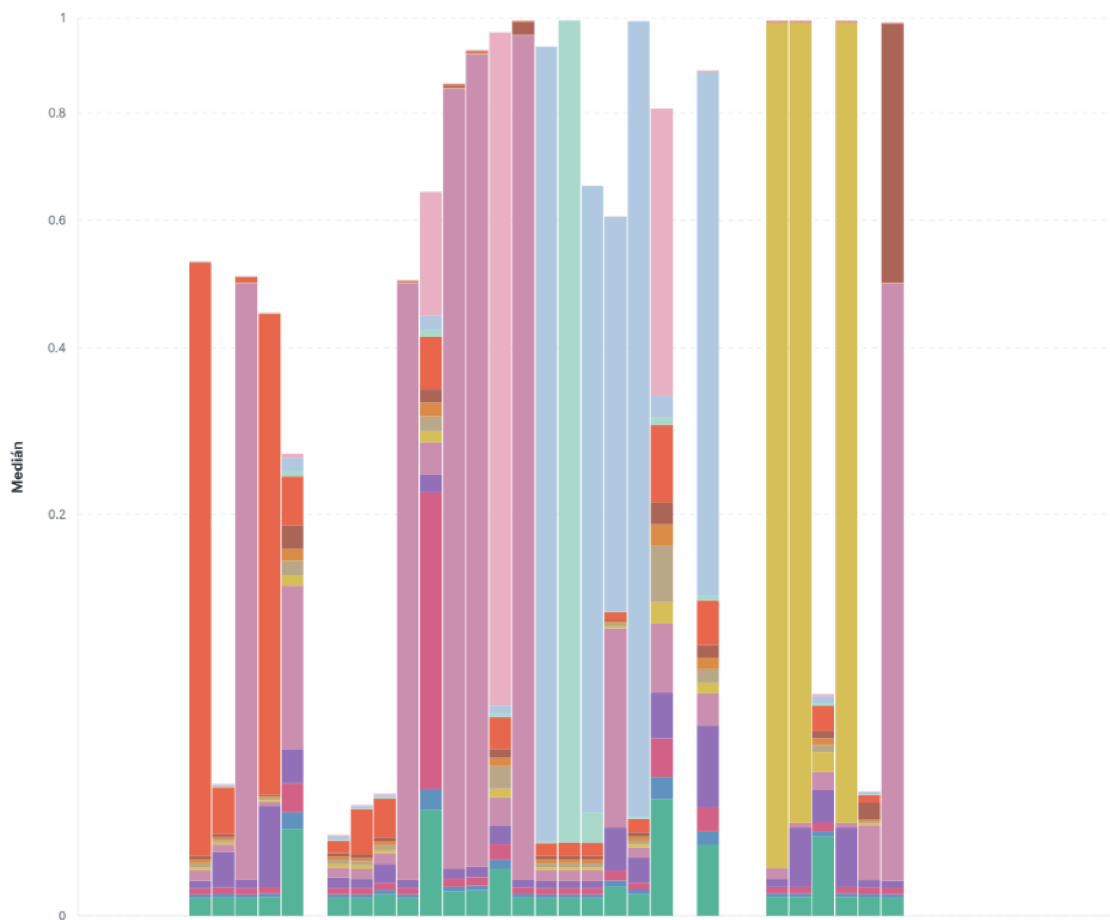
Logstash je nástroj pro sběr, transformaci a zpracování logů a dalších datových toků. Podporuje různé vstupní zdroje dat a umožňuje jejich transformaci pomocí filtrů před jejich uložením nebo odesláním. Díky systému pipeline zvládne mnoho vstupů a výstupů najednou, je tedy horizontálně škálovatelný. Elasticsearch je vyhledávací a analytický nástroj umožňující ukládání, indexování a vyhledávání dat. Je navržen pro vysoký výkon, škálovatelnost a odolnosti vůči selhání. Základem je NoSQL databáze, kterou lze dotazovat pomocí REST API. Kibana je vizualizační nástroj, který poskytuje uživatelské rozhraní nad nástroj Elasticearch. Umožňuje vytvářet různé druhy vizualizací jako jsou grafy, tabulky či dashboardy. Ukázka vizualizace lze vidět na obrázku 4.2.

Správné nasazení a provázání všech tří součástí ELK stacku může být složité, z tohoto důvodu byl použit docker compose[18] ve WSL (Windows Subsytem for Linux).

4.5.4 WSL

Windows Subsystem for Linux[26] je funkce operačního systému Windows, která uživatelům umožňuje pouštět distribuce operačního systému Linux přímo v rámci Windows. Není tak třeba žádného těžkopádného virtuálního stroje či dual bootu. WSL je vytvářeno s myšlenkou hladké a produktivní zkušenosti pro vývojáře, co chtějí používat Windows a Linux najednou. Soubory jsou uložené v separátním Linuxovém souborovém systému, podporuje nástroje příkazové řádky jako BASH, umí pouštět nástroje jako `grep`, `sed` nebo `awk` a další spustitelné ELF-64 programy, a spoustu dalšího.

WSL 2, momentální výchozí verze, využívá virtualizační technologii pro spuštění linuxu ve virtuálním stroji, který je nenáročný na zdroje. Linuxové distribuce běží v izolovaném kontejneru s vlastními procesy, ale mezi instancemi sdílí počítačovou síť, připojené zařízení, hardwarové zdroje a paměť. Hlavním rozdílem druhé verze WSL je vlastní souborový systém, který není napřímo propojen s hostitelským operačním systémem Windows. Zrychluje to tak práci s daty uvnitř WSL, ale značně zpomaluje práci se soubory na hostitelském systému.



Obrázek 4.2: Vizualizace pomocí nástroje Kibana

Kapitola 5

Vyhodnocení

V této kapitole se budu věnovat popisu použitého modelu a jeho výsledkům. Následně vyhodnotím výsledek celkové práce.

5.1 Automatická tvorba datových sad

Pro zjednodušení práce a urychlení iterace byly vytvořeny nástroje na automatický sběr informací pro následné vytvoření trénovací datové sady. Bohužel, některé aplikace se za nejistých okolností odmítly validně spustit, když byly spouštěny ve virtuálním stroji z hostovacího systému. Z tohoto důvodu byly často minimálně reprezentovány v trénování a nemohly tak být správně klasifikované. Důvod tohoto problému je mi nejasný, před i po každém běhu byl virtuální stroj vrácen na identický kontrolní bod.

Použitá datová sada na vyhodnocení byla z tohoto důvodu tvořena částečně ručně. Výsledkem této opravy bylo velmi značné propadnutí přesnosti modelu. V tabulce Skripty, které se měly původně pouštět vzdáleně, byly potřeba pustit přímo na virtuálním stroji. Způsoby jak tomuto problému předejít či ho opravit, mi nejsou jasné. Důvodem může být nejspíš způsob, jak se v takovém prostředí aplikace chová, což je nad rámec této práce.

Před opravou		Po opravě	
Algoritmus	Přesnost	Algoritmus	Přesnost
LightGbmMulti	0,8194	FastForestOva	0,4884
FastTreeOva	0,7336	FastTreeOva	0,4882
FastTreeOva	0,7295	LightGbmMulti	0,4396
LightGbmMulti	0,7164	FastTreeOva	0,3862

Tabulka 5.1: Porovnání výsledků před a po opravě datových sad

5.2 Použitý model

Zvolený model byl výstupem z nástroje ML.NET, který na datové sadě, popsané v předchozí části, trénoval modely po dobu 300 sekund. Bohužel, ML.NET dělá spoustu úprav na pozadí a tak bylo třeba model několikrát přetrénovat, než se dosáhlo přijatelné přesnosti. Výsledky vybraných iterací lze vidět v tabulce 5.2.

Jako výsledný model jsem využil poslední uvedenou iteraci. Model využíval algoritmus FastForestOva, bohužel žádná iterace nepřesáhla hranici přesnosti 0,5. Je možné, že

Iterace 1.		Iterace 2.		Iterace 3.	
Algoritmus	Přesnost	Algoritmus	Přesnost	Algoritmus	Přesnost
FastForestOva	0,4884	LightGbmMulti	0,4396	FastForestOva	0,4884
LightGbmMulti	0,4396	FastTreeOva	0,3862	FastTreeOva	0,4882
FastTreeOva	0,3862	LbfgsLogistic	0,0503	LightGbmMulti	0,4396
FastTreeOva	0,3660	LbfgsMaximum	0,0503	FastTreeOva	0,3862
LbfgsMaxium	0,0503			LbfgsMaximum	0,0503

LbfgsMaximum je zkrácení názvu LbfgsMaximumEntropyMulti
LbfgsLogistic je zkrácení názvu LbfgsLogisticRegressionOva

Tabulka 5.2: Iterace vedoucí k použitému modelu

s využitím robustnější datové sady či delší doby trénování by modely měly vyšší přesnost. Při trénování předem vybraného modelu, který by byl pro mé použití nejoptimálnější, by mohla být teoretická přesnost také vyšší. Model byl trénován na datové sadě obsahující 5307 položek.

5.2.1 Aplikace

Aplikace použité ve virtuálním stroji jsou buď webové browsery a nebo jiné aplikace komunikující po počítačové síti.

Prohlížeče jsem vybral z důvodu snadného a opakovatelného generování síťového provozu. Stačilo pomocí skriptu otevírat přes prohlížeč webové stránky. Zároveň se browsery připojují na celou řadu různých serverů, nemůžeme tak jistě použít TLS handshake rozšíření SNI, které obsahuje doménové jméno serveru. Z prohlížečů jsem vybral Microsoft Edge, Mozilla Firefox, Google Chrome a Opera. Důvodem výběru byla popularita, s těmito prohlížeči se můžete často setkat na počítačích.

Další vybrané aplikace byly také použity z důvodu relativně snadné generace síťového provozu. Šlo primárně o aplikace určených pro komunikaci: Discord, Slack a Telegram. Původně měl být seznam rozšířen ještě o TeamSpeak 3 a Microsoft Teams. U těchto aplikací bohužel nastal problém v automatizovaném spouštění přes skript. Kromě aplikací na komunikaci byly použity aplikace Steam a Spotify, obě také generují síťový provoz bez větších obtíží.

5.3 Výsledky klasifikace

Ověření modelu probíhalo ručním pouštěním aplikací ve virtuálním stroji zatímco byl její provoz sledován nástrojem Suricata. Obsah výstupního souboru `eve.json` byl následně pomocí nástroje Fluent Bit přeposílán do klasifikátoru, který klasifikovaná data posílal do ELK stacku. Z nástroje Kibana (vizualizační část ELK stacku) také pochází veškeré grafy v této kapitole.

Na virtuálním stroji byly postupně pouštěny po dobu deseti minut webové prohlížeče otevírající náhodné stránky. Zde jsem chtěl primárně ověřit, zda li model zvládne klasifikovat jednotlivé prohlížeče od sebe. Ostatní aplikace jsem nechal běžet po dobu pěti minut a ručně v nich vytvářel síťový provoz. Ať už se jednalo o připojování se na různé servery či psaní zpráv. Spojení, které tyto aplikace vytvářeli, nemusí být tedy opakovatelnými.

Google Chrome, Microsoft Edge i Opera používají Chromium jako dvé jádro. Framework Electron, na kterém je stavěn Discord a Slack, také používá Chromium jako své jádro, Steam a Spotify používaji Chromium Embedded Framework místo Electronu. Skoro v každém případě se pohybujeme v skoro identických jádrech aplikací a může tak být složité je od sebe čistě pomocí síťového toku rozeznat.

5.3.1 Výsledky

Výsledky ve většině případů vyvrátili moji původní domněnku o nerozlišitelnosti Chromium aplikací. Většina aplikací buď šla rozeznat díky doménovému jménu serveru, které bylo získáno z rozšíření SNI a nebo pomocí jejich šifer a rozšíření posílanými v TLS spojení. První varianta, SNI, hrála podstatně větší roli u aplikací, které se budou pravidelně připojovat na stejné adresy. U prohlížečů byla tato hodnota nepředvídatelná, ale jejich sady rozšíření a šifer byly hlavním identifikátorem.

Osa X je čas, každý sloupeček odpovídá jedné minutě. Osa Y je zobrazena na odmocninové škále, lze tak vidět jak nízké tak vyšší hodnoty najednou. Hodnoty jsou mediánem hodnot v daném intervalu. Medián mi dovolí zobrazit hodnotu při jakémkoli množství záznamů v danou dobu a na rozdíl od průměru není výrazně snižován dalšími spojeními probíhající ve stejnou dobu.

Prohlížeče

V této části se budu věnovat výsledku prohlížečů. Hodnoty Google Chrome jsou v zelené, Opera v fialové, Mozilla Firefox v červené a Microsoft Edge v modré. Počet záznamů je v oranžové.

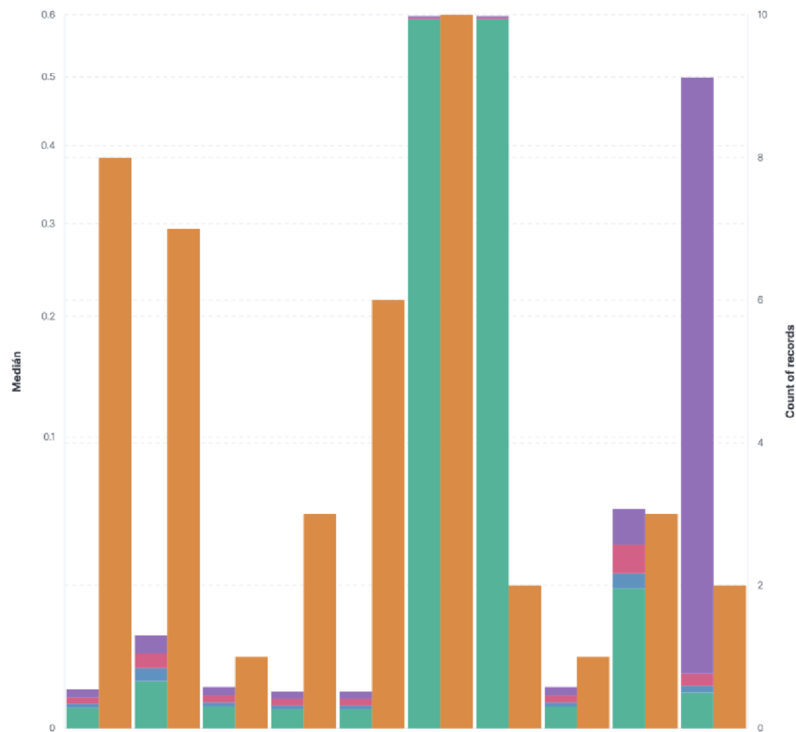
U Google Chrome vidíme na obrázku 5.1, že si můžeme být detekcí tohoto prohlížeče relativně jistí. I když nejvyšší hodnoty mediánu dosahují pouze pod 0,6 tak se málokdy kryje s dalšími prohlížeči či aplikacemi. Problematické může být malé množství detekovaných spojení, kde jich bylo v rámci jednotek.

Microsoft Edge už zas tak rozeznatelný není, jak lze vidět na obrázku 5.2. Ve většině případů je zaměňován za prohlížeč Opera. Správnou klasifikaci vidíme pouze ve druhém sloupečku a v posledním.

Mozilla Firefox je na rozdíl od předchozích velmi jasně rozpoznatelným prohlížečem, jak lze vidět na obrázku 5.3. Důvodem bude velmi odlišný JA4 hash TLS spojení. Zatímco ostatní prohlížeče je mají velmi podobné, až identické, Firefox je má odlišné, jak lze vidět v tabulce 5.3. Firefox také na rozdíl od Chrome a Edge produkoval velké množství spojení, až nižší stovky.

Prohlížeč	JA4_a	JA4_b
Firefox	t13d1715xx	5b57614c22b0
Edge	t13d1516xx	8daaf6152771
Chrome	t13d1516xx	8daaf6152771
Opera	t13d1516xx	8daaf6152771

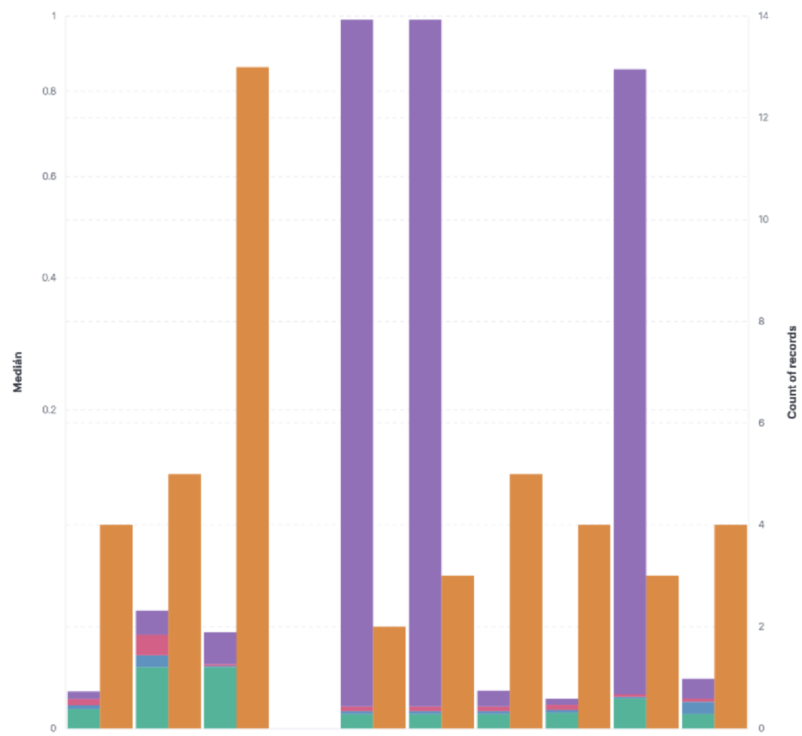
Tabulka 5.3: JA4 hashe prohlížečů



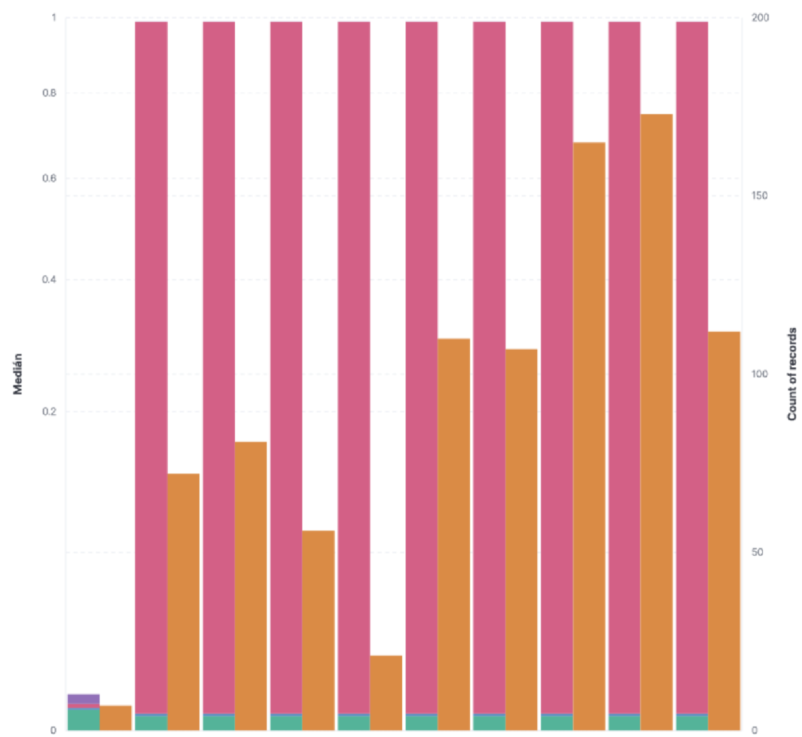
Obrázek 5.1: Klasifikace při používání prohlížeče Google Chrome

Prohlížeč opera je také značně jednoznačný, jak lze vidět na obrázku 5.4. Jak je ale vidět v tabulce 5.3, má identické JA4 hashe jako Edge a Chrome. Důvodem této jednoznačnosti je nejspíš množství spojení. Zatímco u Edge a Chrome byly zachyceny jednotky spojení, u Opery se znovu pohybujeme až v nižších desítkách.

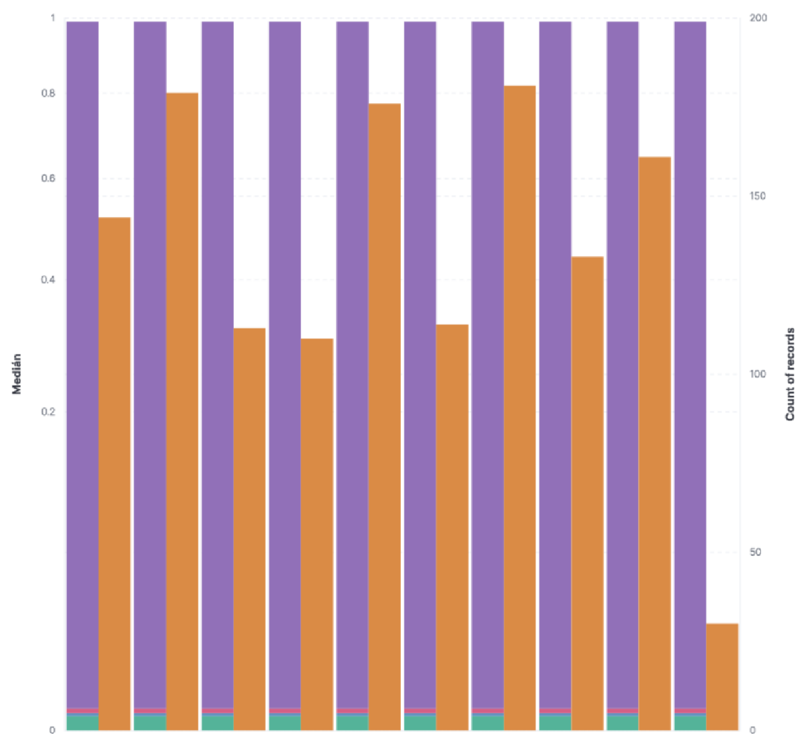
Ostatní aplikace jsou také snadno rozpoznatelné, jak lze vidět v 5.5. Barevně, Spotify v zelené, Steam v růžové, TeamSpeak 3 v modré, Discord v hnědé a Slack tmavě oranžová. Lze zde vidět i kolik jaká aplikace tvoří spojení v rámci používání.



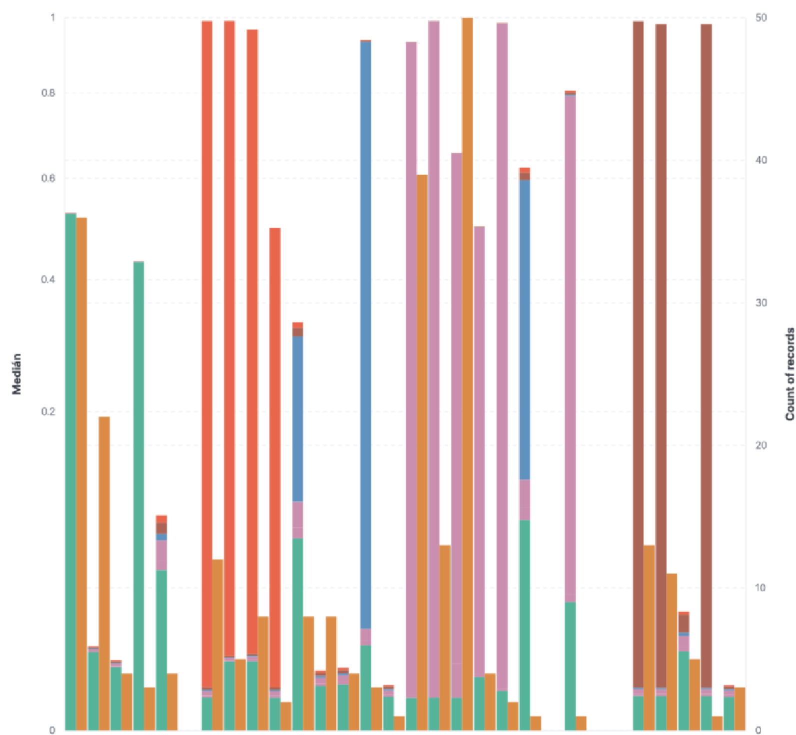
Obrázek 5.2: Klasifikace při používání prohlížeče Microsoft Edge



Obrázek 5.3: Klasifikace při používání prohlížeče Mozilla Firefox



Obrázek 5.4: Klasifikace při používání prohlížeče Opera



Obrázek 5.5: Klasifikace při použití dalších aplikací

Kapitola 6

Závěr

Detekce aplikací ze šifrované komunikace podle TLS handshake je možné v určitých případech. V práci jsem využil části otisku JA4, která lze spočítat z JA3 string, který není zatím plně podporován v nástrojích na sledování síťového provozu.

Detekování za pomocí strojového učení vyžaduje mít datovou sadu obsahující dostatečné množství záznamů pro natrénování. Toto byl také důvod proč prohlížeče Microsoft Edge a Google Chrome byly častokrát chybně klasifikovány jako prohlížeč Opera. Počet záznamů o spojení prohlížeče Opera byl řádově vyšší než u dalších. Případně je nutné, aby aplikace měla originální JA4 hash, jako například Mozilla Firefox.

Detekovat lze i jiné aplikace než prohlížeče, jakokomunikační či herní aplikace. Zde je možné více použít pro klasifikaci i adresu vzdáleného serveru, která se pravděpodobně nebude u takovýchto aplikací pravidelně měnit.

Problematikou stále zůstává nutnost mít datovou sadu, a tak pro monitorování sítě bude lepší detekovat anomálie. Otisky JA4 filtrují pryč hodnoty GREASE a zároveň řadí hodnoty šifer a rozšíření. Tímto je schopen udržet větší konzistentnost pro danou aplikaci i přes několik spojení.

Literatura

[1] *CVE-2011-3389* [online]. 2011 [cit. 2024-04-29]. Dostupné z: https://www.cve.org/CVERecord?id=CVE-2011-3389.

[2] *CVE-2014-3566* [online]. 2014 [cit. 2024-04-29]. Dostupné z: https://www.cve.org/CVERecord?id=CVE-2014-3566.

[3] Allen, C. a Dierks, T. *The TLS Protocol Version 1.0* [RFC 2246]. RFC Editor, leden 1999. DOI: 10.17487/RFC2246. Dostupné z: https://www.rfc-editor.org/info/rfc2246.

[4] Althouse, J. B., Atkinson, J. a Atkins, J. *Open sourcing JA3* [online]. 2017 [cit. 2024-04-29]. Dostupné z: https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41/.

[5] Althouse, J. *JA4+ Network Fingerprinting* [online]. 2023 [cit. 2024-04-29]. Dostupné z: https://blog.foxio.io/ja4+-network-fingerprinting.

[6] Althouse, J. *JA4+ technical details* [online]. 2023 [cit. 2024-04-30]. Dostupné z: https://github.com/FoxIO-LLC/ja4.

[7] Anderson, B., Chi, A., Dunlop, S. a McGrew, D. *Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption* [online]. arXiv, 2018. Dostupné z: https://arxiv.org/abs/1805.115440.

[8] Benjamin, D. *Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility* [RFC 8701]. RFC Editor, leden 2020. DOI: 10.17487/RFC8701. Dostupné z: https://www.rfc-editor.org/info/rfc8701.

[9] Cloudflare. *Cloudflare Radar - Adoption & Usage* [online]. 2024 [cit. 2024-04-29]. Dostupné z: https://radar.cloudflare.com/adoption-and-usage.

[10] Dierks, T. a Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.1* [RFC 4346]. RFC Editor, duben 2006. DOI: 10.17487/RFC4346. Dostupné z: https://www.rfc-editor.org/info/rfc4346.

[11] Freier, A. O., Karlton, P. a Kocher, P. C. *The Secure Sockets Layer (SSL) Protocol Version 3.0* [RFC 6101]. RFC Editor, srpen 2011. DOI: 10.17487/RFC6101. Dostupné z: https://www.rfc-editor.org/info/rfc6101.

[12] IBM. *What Is Random Forest?* [online]. [cit. 2024-05-08]. Dostupné z: https://www.ibm.com/topics/random-forest.

[13] Iyengar, J. a Thomson, M. *QUIC: A UDP-Based Multiplexed and Secure Transport* [RFC 9000]. RFC Editor, květen 2021. DOI: 10.17487/RFC9000. Dostupné z: https://www.rfc-editor.org/info/rfc9000.

[14] Linder, B. *Differences between Windows 11, Home, Pro, SE, Enterprise, and Education* [online]. 2021 [cit. 2024-05-05]. Dostupné z: https://liliputing.com/differences-between-windows-11-home-pro-enterprise-and-education/.

[15] Liu, C., Han, J. a Wei, Q. *Browser Identification Based on Encrypted Traffic* [online]. Atlantis Press, 2016/09. DOI: 10.2991/cimns-16.2016.90. ISSN 2352-538X. Dostupné z: https://doi.org/10.2991/cimns-16.2016.90.

[16] Maskosyan, S. *TLS vs. SSL: Evolution, Differences, and Best Practices* [online]. 2024 [cit. 2024-04-29]. Dostupné z: https://10web.io/blog/tls-vs-ssl-evolution-differences-and-best-practices/.

[17] Microsoft. *LightGBM documentation* [online]. 2023 [cit. 2024-04-29]. Dostupné z: https://lightgbm.readthedocs.io/en/stable/.

[18] Mitchell, E. *Getting started with the Elastic Stack and Docker Compose: Part 1* [online]. 2023 [cit. 2024-05-08]. Dostupné z: https://www.elastic.co/blog/getting-started-with-the-elastic-stack-and-docker-compose.

[19] Mockapetris, P. *Domain names - concepts and facilities* [RFC 1034]. RFC Editor, listopad 1987. DOI: 10.17487/RFC1034. Dostupné z: https://www.rfc-editor.org/info/rfc1034.

[20] Mockapetris, P. *Domain names - implementation and specification* [RFC 1035]. RFC Editor, listopad 1987. DOI: 10.17487/RFC1035. Dostupné z: https://www.rfc-editor.org/info/rfc1035.

[21] Mozzila. *Mozilla telemetry public data on ssl ratios* [online]. [cit. 2023-01-14]. Dostupné z: https://public-data.telemetry.mozilla.org/api/v1/tables/telemetry_derived/ssl_ratios/v1/files.

[22] Nielsen, H., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J. et al. *Hypertext Transfer Protocol – HTTP/1.1* [RFC 2616]. RFC Editor, červen 1999. DOI: 10.17487/RFC2616. Dostupné z: https://www.rfc-editor.org/info/rfc2616.

[23] Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3* [RFC 8446]. RFC Editor, srpen 2018. DOI: 10.17487/RFC8446. Dostupné z: https://www.rfc-editor.org/info/rfc8446.

[24] Rescorla, E. a Dierks, T. *The Transport Layer Security (TLS) Protocol Version 1.2* [RFC 5246]. RFC Editor, srpen 2008. DOI: 10.17487/RFC5246. Dostupné z: https://www.rfc-editor.org/info/rfc5246.

[25] Thomson, M. a Turner, S. *Using TLS to Secure QUIC* [RFC 9001]. RFC Editor, květen 2021. DOI: 10.17487/RFC9001. Dostupné z: https://www.rfc-editor.org/info/rfc9001.

[26] WOJCIAKOWSKI, M., HAMMONS, J., LOEWEN, C. a GEÇDOĞAN, F. *What is the Windows Subsystem for Linux?* [online]. 2023 [cit. 2024-05-08]. Dostupné z: https://learn.microsoft.com/en-us/windows/wsl/about.

[27] YATES, D. a ISLAM, M. Z. FastForest: Increasing random forest processing speed while maintaining accuracy. *Information Sciences*. 2021, sv. 557, s. 130–152. DOI: https://doi.org/10.1016/j.ins.2020.12.067. ISSN 0020-0255. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0020025520312330.

Příloha A

Datová sada

V této příloze je ukázka datové sady použité na trénování finální verze modelu. V přiložených souborech pojmenována `dataset.csv`

Flow_id	Dest_ip	Src_port	Tls_sni
1716980329334140	20.190.177.146	50830	login.microsoftonline.com
40893768016259	52.113.194.132	50841	config.teams.microsoft.com
451310318498404	2.22.112.38	50858	kv601.prod.do.dsp.mp.microsoft.com
496102113060057	23.62.220.32	50859	cp601.prod.do.dsp.mp.microsoft.com
541048805324883	52.245.136.46	50852	pf.pipe.aria.microsoft.com
346687200726753	52.182.143.213	50851	teams.events.data.microsoft.com
378841782626250	52.168.117.171	50857	us-teams.events.data.microsoft.com
393830667661752	20.140.137.181	50855	tb.pipe.aria.microsoft.com
577199770310433	20.54.25.16	50870	array617.prod.do.dsp.mp.microsoft.com
911834388660087	2.18.69.84	50878	oneclient.sfx.ms
2098595344036050	13.69.239.74	50882	eu-mobile.events.data.microsoft.com

Tabulka A.1: První část datové sady

Tls_version	Tls_ja4_a	Tls_ja4_b	Tag
TLS 1.3	t13d2012xx	2b729b4bf6f3	lsass
UNDETERMINED	t13d2011xx	2b729b4bf6f3	ms-teamsupdate
UNDETERMINED	u13d1809xx	4b22cbed5bed	svchost
UNDETERMINED	u13d1809xx	4b22cbed5bed	svchost
TLS 1.3	t13d2011xx	2b729b4bf6f3	ms-teamsupdate
TLS 1.3	t13d2011xx	2b729b4bf6f3	ms-teamsupdate
TLS 1.3	t13d2011xx	2b729b4bf6f3	ms-teamsupdate
TLS 1.3	t13d2011xx	2b729b4bf6f3	ms-teamsupdate
TLS 1.2	t12d1809xx	4b22cbed5bed	svchost
UNDETERMINED	t13d2012xx	2b729b4bf6f3	svchost
TLS 1.3	t13d2012xx	2b729b4bf6f3	OneDrive
UNDETERMINED	t13d1516xx	8daaf6152771	Spotify

Tabulka A.2: Druhá část datové sady